Q1 sequence detector

State Transition Diagram

首先先定義好在每個 state 時所偵測到的 input 會對應到哪一個 next_state 及 output。

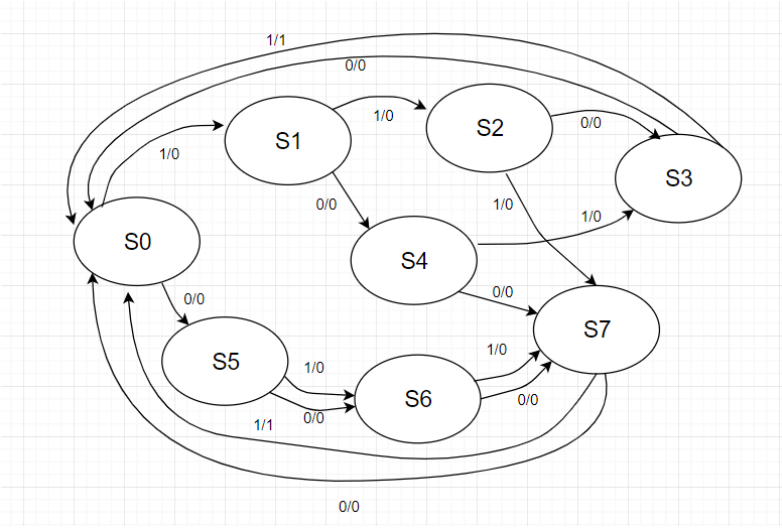
第一步先做出能順利 1101 的功能，首先令初始的 state 為 S0，當 rst 或重新偵測一個數列時會從 S0 開始，若成功接受到 1，則會跳到 S1，output 0；再成功接受到 1 會進入 S2，output 0；再成功接受到 0，會跳到 S3，再 S3 時，若 input 為 1，output 為 1，若 input 為 0，output 為 0，下一個 state 都會回到 S0。

接著做出能順利接收 1011 的功能，於第一步要偵測 1，和偵測 1101 時的情況一樣，所以在這階段我們一樣將 S0 作為其始點，若成功接收到 1 則進入 S1，output 0；而在 S1 時，我們所要偵測的數字是 0，因此定義若是接受到 0 會進入到 S4，output 0，若否，一樣進入 S2 做出偵測 1101 的功能；而在 S4 時，若偵測到 0，下一步要偵測 1，若是偵測到 1 須 output 1，這點的功能與 S3 相同，因此若成功接收到 0，進入 S3，output 0。

處理完成功偵測 1101 及 1011 的功能後，開始處理若是確定不會偵測測數列時的情況，我們定義三個 state，此三個 state 為確定不會偵測到數列時所會跳到的 state，另 S5 會跳到 S6，S6 會跳到 S7，S7 會跳回 S0。在 S0 時，若偵測到 0，會跳到 S5，在 S5 階段之後會偵測第二位數字跳到 S6，接著在 S6 會偵測第三位數字跳到 S7，而在 S2 時若是沒有偵測到 1，會進入 S7 偵測完最後一位數字，在 S4 時，若是沒有偵測到 1，會進入 S7 偵測最後一位數字，而最後 S7 偵測完數字之後就會跳回 S0，偵測下一組數字。

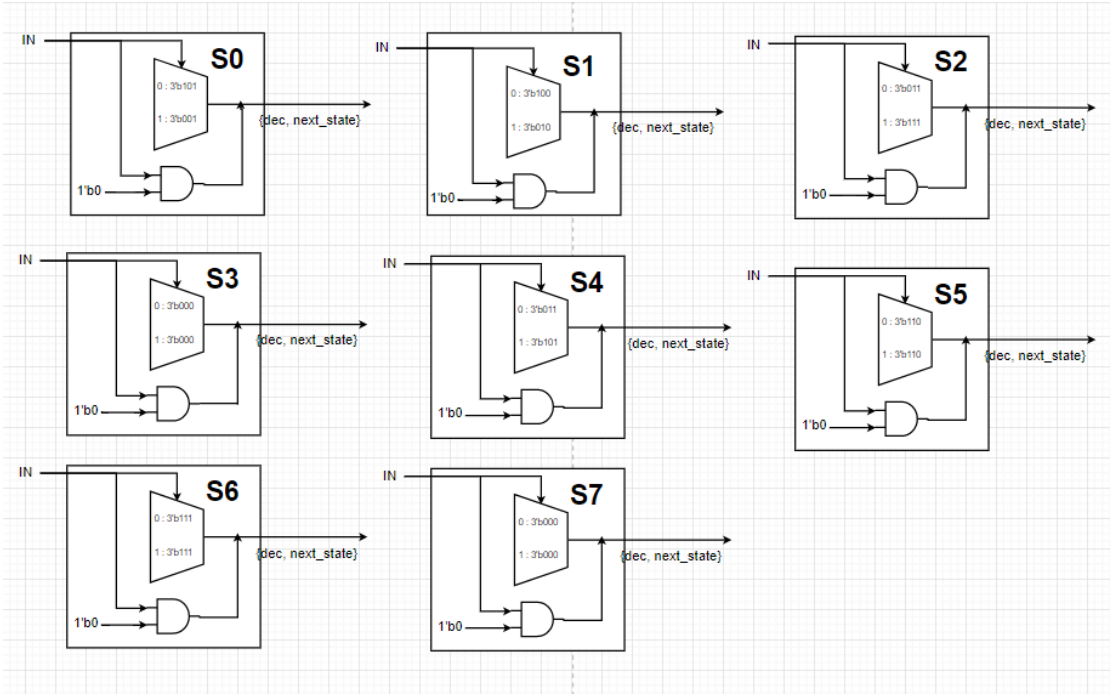
將此三種情況處理完之後，就將所有會發生的情況都解決了。以下為完整的

State Transition Diagram。

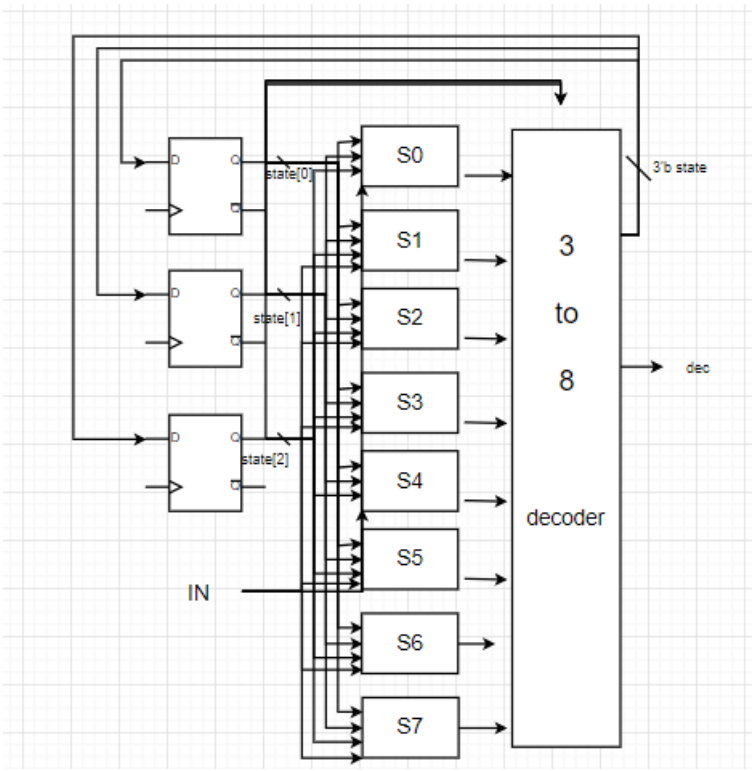


Logic Circuit

首先我們先將每一個 STATE 的邏輯電路實做出來，以下為每一個 state 的邏輯電路圖。

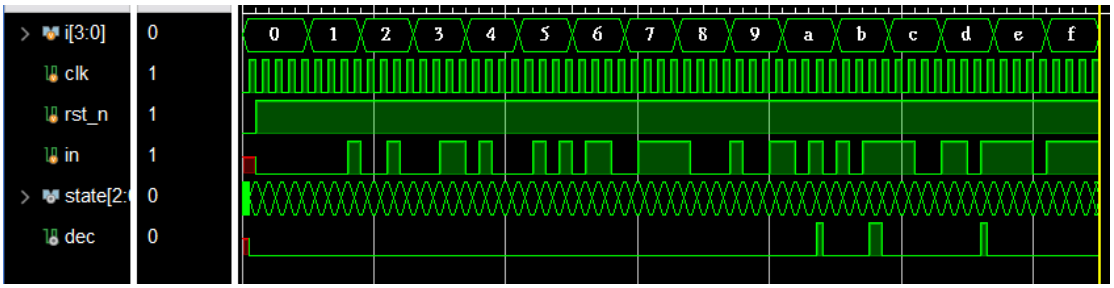


接著把每一個 block 接上去合成最終的邏輯電路。



Verification

利用 verilog 驗證邏輯電路是否為正確 Testbench 設計方式 將每組 4bit 的組合都跑過一次，驗證是否只有在 input = 4’b1101 或 4’b1011 時 output 會等 於 1，且若 1101 連續但 1101 在不同組的數字上(如 : 00110100)，結果要為錯誤。



我們能觀察到當 input 為 1010 時，在 S3 的 output 會被拉起來半個週期，這是因為在 S3 的前半個週期時所讀取的 input 仍是第三個 bits，因此 output 會短暫輸出一，但在後半個週期讀到最後一個數字時又變回 0，因此仍符合 mealy machine 的特性，扣除此點之後，只有在 input 為 4’b1101 會 4’b1011 時會輸出 1，因此可以證明此邏輯電路為正確的。

Question2

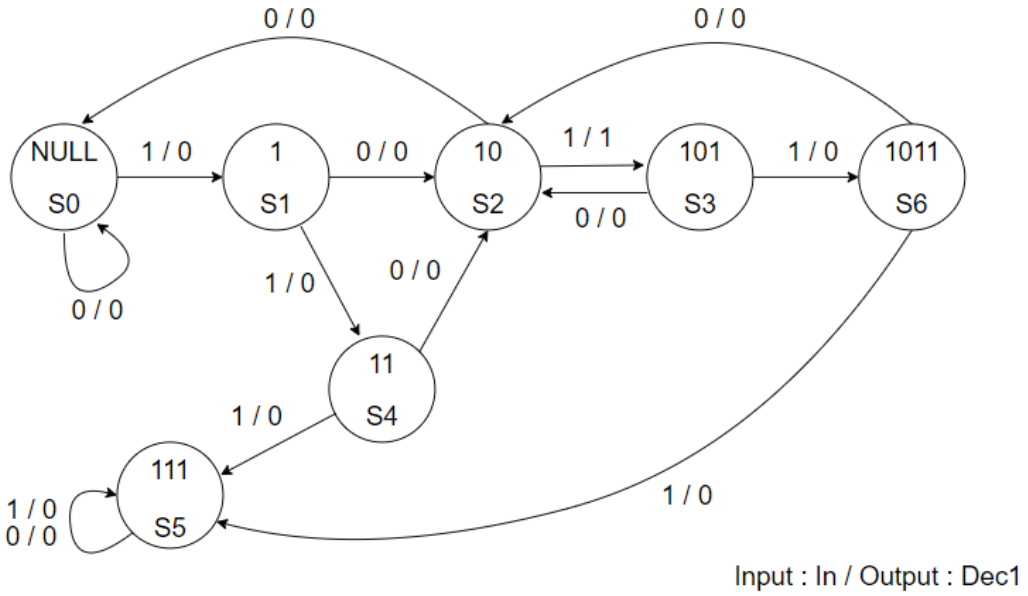
首先，首先先定義好在每一個 state，然後再畫出此題的 State Transition Diagram，如下：

State：S0, S1, S2, S3, S4, S5, S6；(for Dec1)

state：s0, s1, s2, s3, s4；(for Dec2)

Input：In, rst_n；

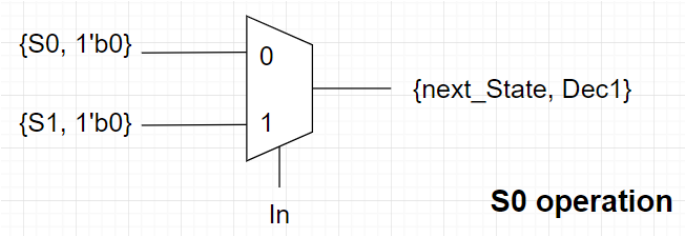
Output：Dec1, Dec2；



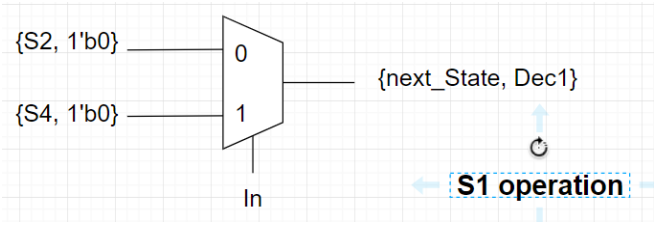
For Dec1(detect 101):

一開始 rst_n = 1'b1 時，還未進入任何 state；

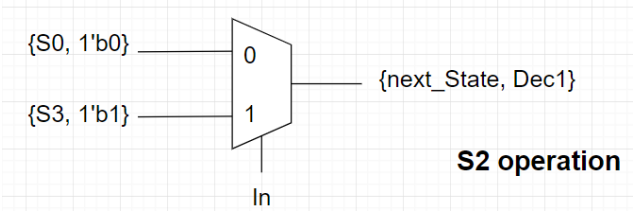
當 rst_n = 1'b0 時，開始進入 S0(State0 為還沒有任何數字進去，NULL 的情形)，當 In = 0 / Dec1 = 0，next_State = S0(一開始輸入的數字為 0 相當於是 NULL 的情形)；當 In = 1 / Dec1 = 0，next_State = S1(State1 代表 sequence 為 1)。



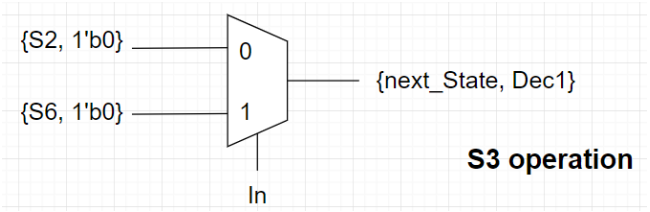
進入到 S1 後(sequence 為 1)，當 In = 0 / Dec1 = 0，next_State = S2(State2 代表 sequence 為 10)；當 In = 1 / Dec1 = 0，next_State = S4(State4 代表 sequence 為 11)。



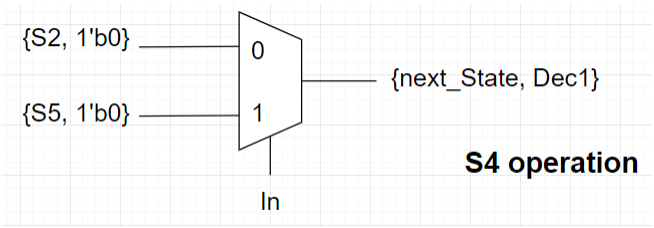
進入到 S2 後(sequence 為 10)，當 In = 0 / Dec2 = 0，next_State = S0(sequence 為 100，相當於 NULL 的情形)；當 In = 1 / Dec2 = 1，next_State = S3(State3 代表 sequence 為 101)。



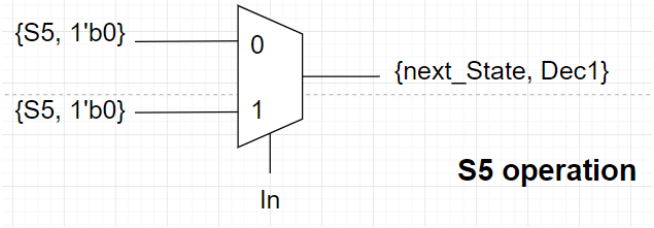
進入到 S3 後(sequence 為 101)，當 In = 0 / Dec1 = 0，next_State = S2(sequence 為 1010，相當於 sequence 為 10 的情形，因此為 State2)；當 In = 1 / Dec1 = 0，next_state = S6(State6 代表 sequence 為 1011)。



進入到 S4 後(sequence 為 11)，當 In = 0 / Dec1 = 0，next_State = S2(sequence 為 110，相當於 sequence 為 10 的情形，因此為 State2)；當 In = 1 / Dec1 = 0，next_state = S5(State5 代表 sequence 為 111)。

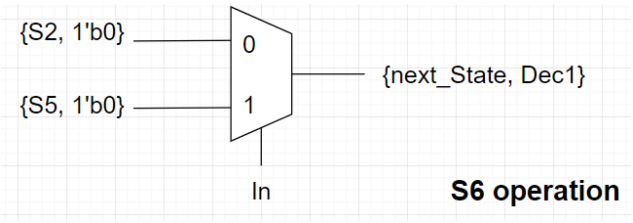


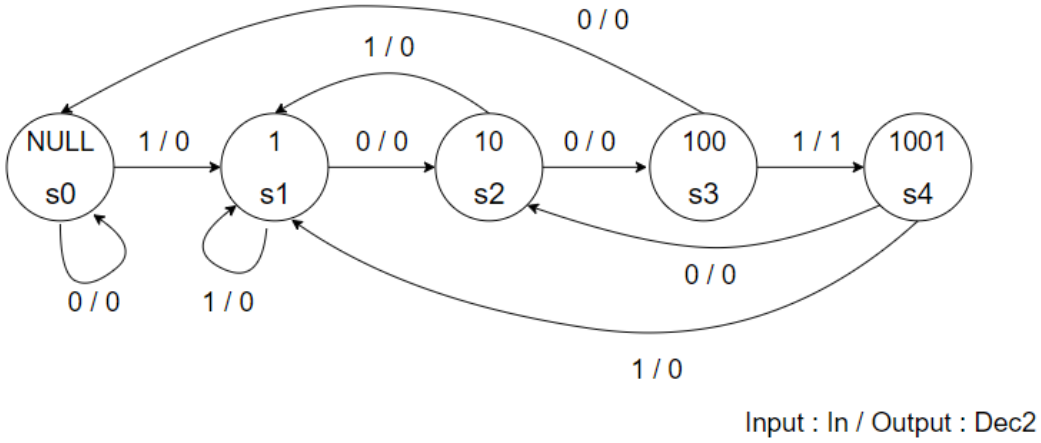
進入到 S5 後(sequence 為 111)，當 In = 0 / Dec1 = 0 或 In = 1 / Dec1 = 0，next_State = S5(spec：Dec1 == 1'b1 when input is 101 AND no 111 occurs before)。



進入到 S6 後(sequence 為 1011)，當 In = 0 / Dec1 = 0，next_State = S2(sequence 為 10110，相當於 sequence 為 10 的情形，因此為 State2)；當 In = 1 / Dec1 =

0，next_State = S5(sequence 為 10111，相當於 sequence 為 111 的情形，因此為 State5)。

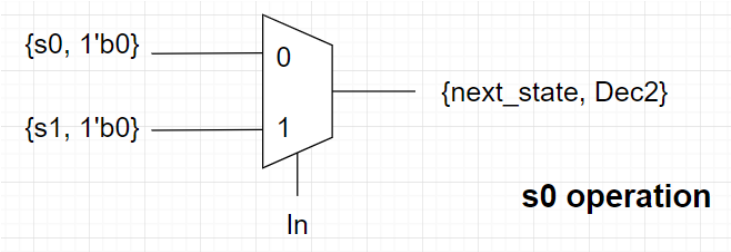




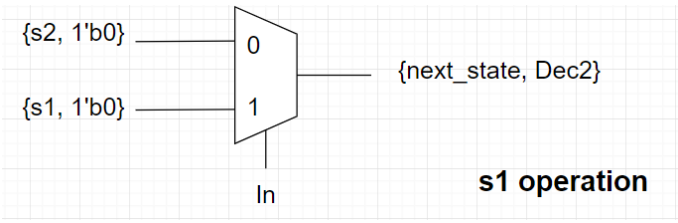
For Dec2(detect 1001):

一開始 rst_n = 1’b1 時，還未進入任何 state；

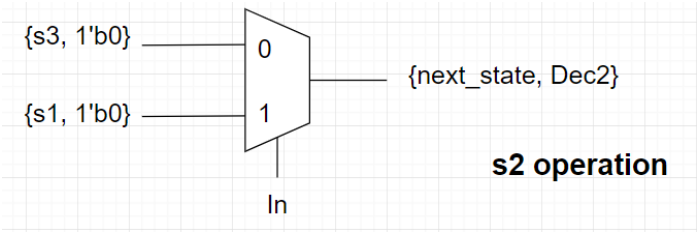
當 rst_n = 1’b0 時，開始進入 s0(State0 為還沒有任何數字進去，NULL 的情形)，當 In = 0 / Dec2 = 0，next_state = s0(一開始輸入的數字為 0 相當於是 NULL 的情形)；當 In = 1 / Dec2 = 0，next_state = s1(state1 代表 sequence 為 1)。



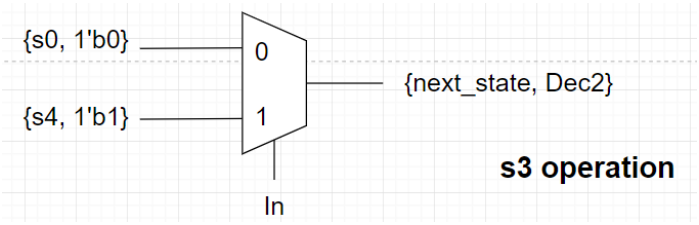
進入到 s1 後(sequence 為 1)，當 In = 0 / Dec2 = 0，next_state = s2(state2 代表 sequence 為 10)；當 In = 1 / Dec2 = 0，next_state = s1(sequence 為 11，相當於 sequence 為 1 的情況)。



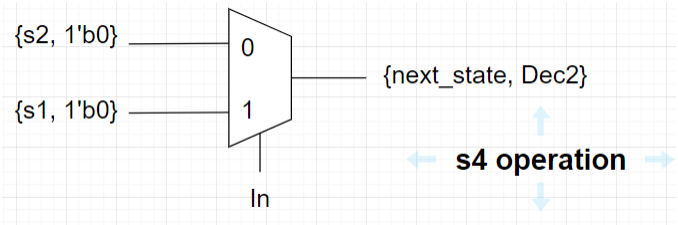
進入到 s2 後(sequence 為 10)，當 In = 0 / Dec2 = 0，next_state = s3(state3 代表 sequence 為 100)；當 In = 1 / Dec2 = 0，next_state = s1(sequence 為 101，相當於 sequence 為 1 的情況，因此為 state1)。



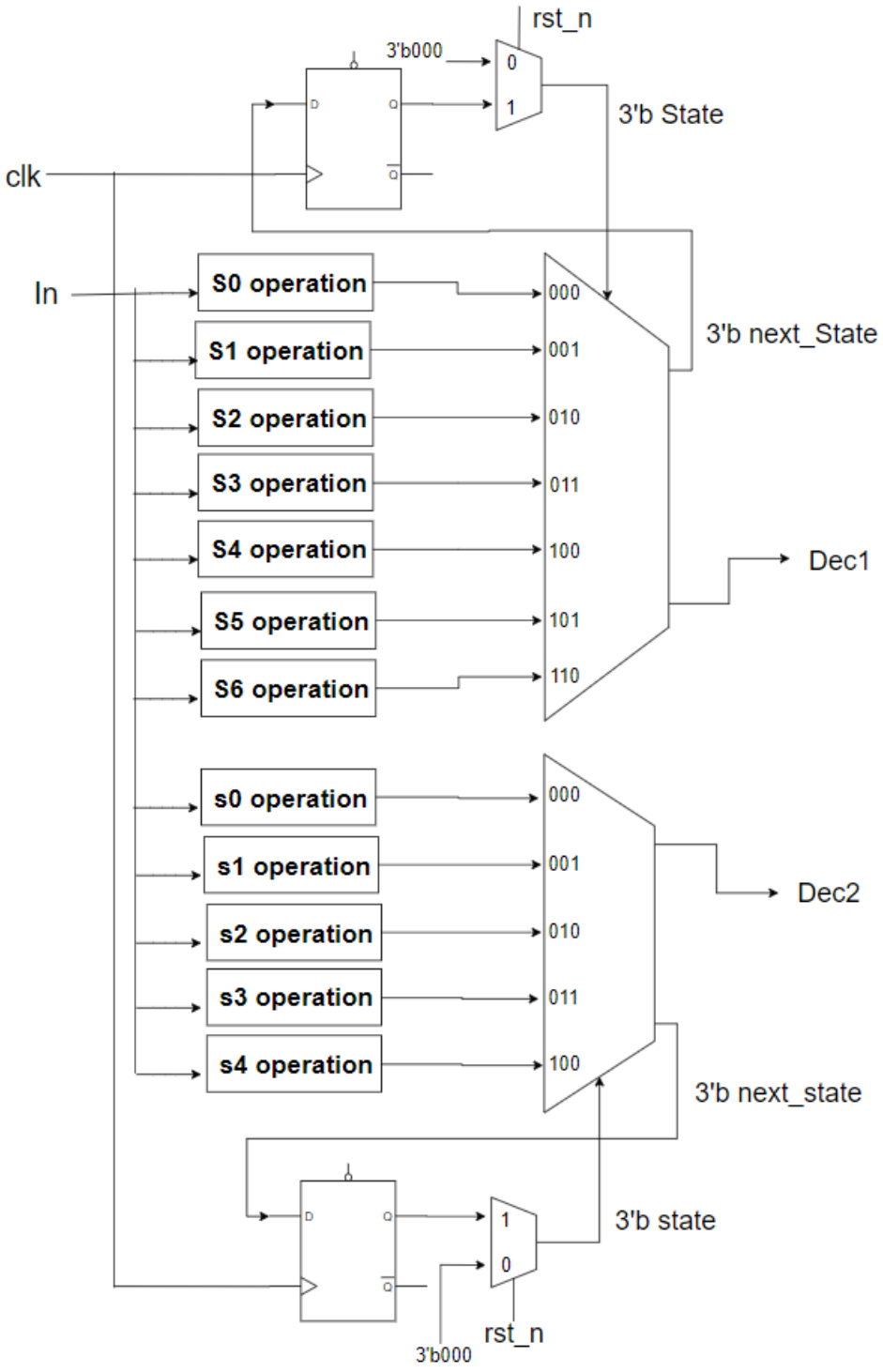
進入到 s3 後(sequence 為 100)，當 In = 0 / Dec1 = 0，next_state = s0(sequence 為 1000，相當於 NULL 的情形，因此為 state0)；當 In = 1 / Dec1 = 0，next_state = s4(state4 代表 sequence 為 1001)。



進入到 s4 後(sequence 為 1001)，當 In = 0 / Dec1 = 0，next_state = s2(sequence 為 10010，相當於 sequence 為 10 的情形，因此為 state2)；當 In = 1 / Dec1 = 0，next_state = s1(sequence 為 10011，相當於 sequence 為 1 的情形，因此為 state1)。

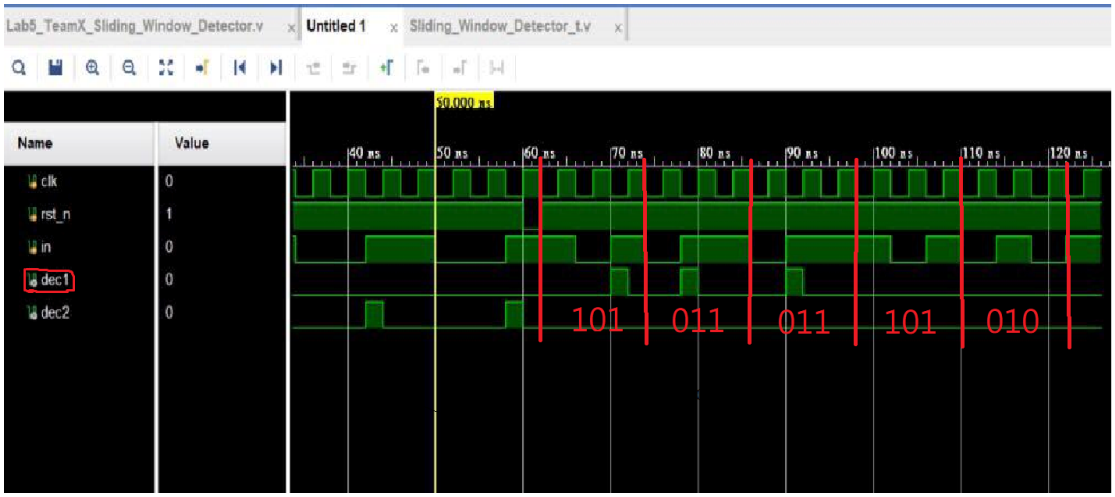


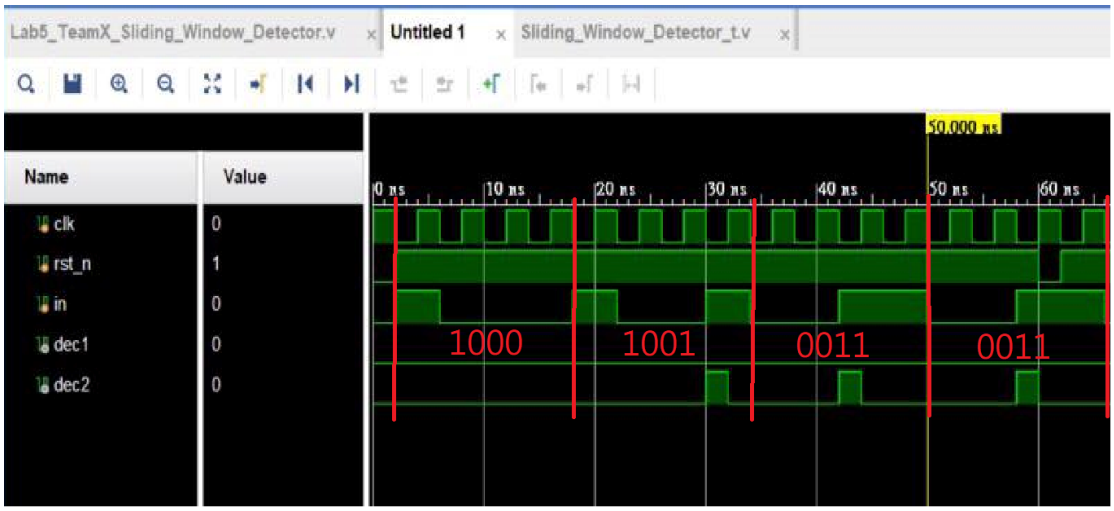
以下是完整的電路設計圖：


rst_n
3'b000
0
1
3'b State
clk
In
S0 operation
S1 operation
S2 operation
S3 operation
S4 operation
S5 operation
S6 operation
000
001
010
011
100
101
110
3'b next_State
Dec1
s0 operation
s1 operation
s2 operation
s3 operation
s4 operation
000
001
010
011
100
Dec2
3'b next_state
1
0
3'b state
3'b000
rst_n

電路檢測

以下是 vivado 跑的波形模擬：





此題的電路檢測方式為檢測一些特定 case 的 sequence，再觀察 vivado 的波形模擬圖是否正確。

由以上的波形模擬圖可知：

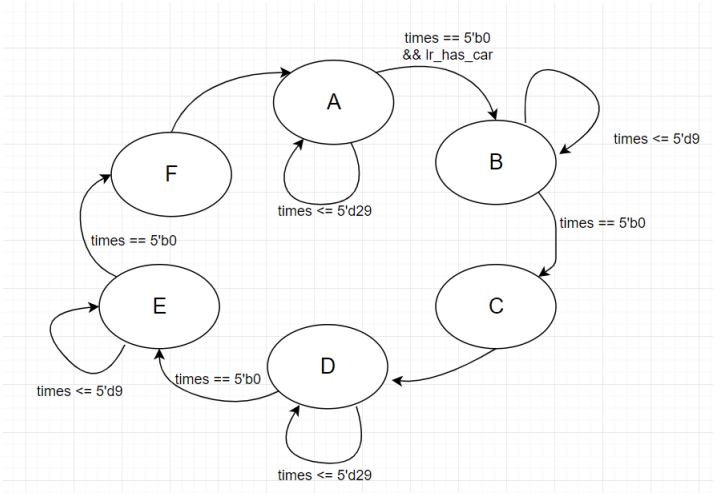
首先先觀察第一張圖(For Dec1)，一開始有偵測到 101，所以 Dec1 = 1；接著的 sequence 為 011，因為前一個 sequence 為 101，因此合起來看為 101011，當中又有一個 101 的 sequence，所以 Dec1 = 1；然而當中間開始出現 111 的 sequence 時，後面就算有出現 101 的 sequence，Dec1 = 0，符合題幹所要求。

接著再觀察第二張圖(For Dec2)，一開始的 sequence 為 1000，所以 Dec2 = 0；接著的 sequence 為 1001，所以 Dec2 = 1；然而在下一個 sequence 為 0011，因為前一個 sequence 為 1001，合起來看為 1001001，當中又有一個 1001 的 sequence，所以 Dec2 = 1，以上皆符合此題所要求。

Q3 traffic light controller

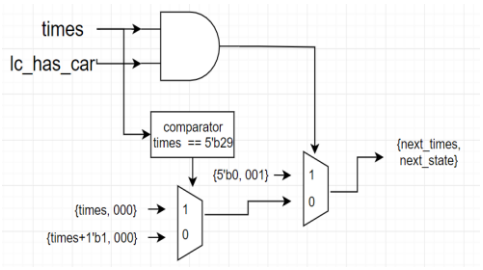
State Transition Diagram

首先先定義好在每個 state 時所偵測到的 input 或 reg 信號會對應到哪一個 next_state 及 output。在此需要六個 state、一個 5 – bits register(times)訊號紀錄時間及當作變換 state 的訊號。

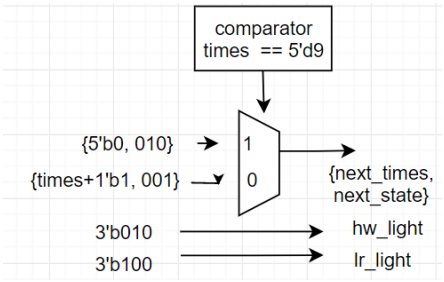


Logic Circuit

第一個是 state A，代表 HW 是綠燈而 LR 是紅燈，hw_light 為 3’b001，lr_light 為 3’b100。在此 state 時，若是時間還沒數到 29 或 lr_has_car == 1’b0，next_state 一樣維持在 state_A，若 times 數到 29 且 lr_has_car == 1’b1，next_state 就會轉換到 state B，左下為 state_A 邏輯電路圖。



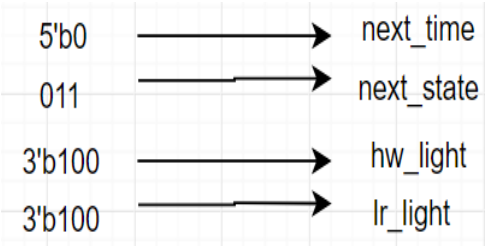
State A

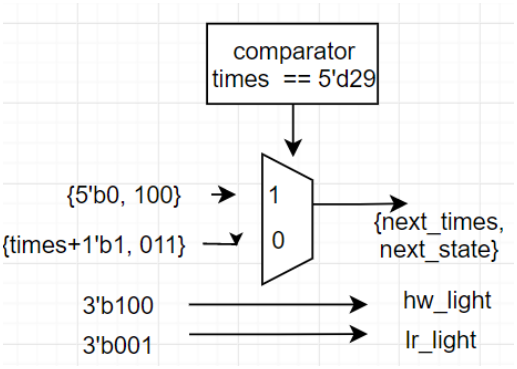


State B

下一個是 state B，代表 HW 是黃燈而 LR 是紅燈，output hw_light 為 3’b010，lr_light 為 3’b100。在此 state 時，若是時間還沒數到 9，next_state 一下一個 state 是 C，代表 HW 和 LR 都是紅燈，output hw_light 及 lr_light 皆為 3’b100。在此 state 時，經過一個 cycle 就會跳到 state D，next_time 設為 0，左下為 state C 的邏輯電路圖。

下一個 state 是 C，代表 HW 和 LR 都是紅燈，output hw_light 及 lr_light 皆為 3’b100。在此 state 時，經過一個 cycle 就會跳到 state D，next_time 設為 0，左下為 state C 的邏輯電路圖。

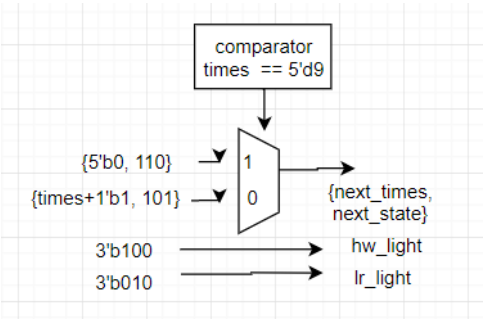


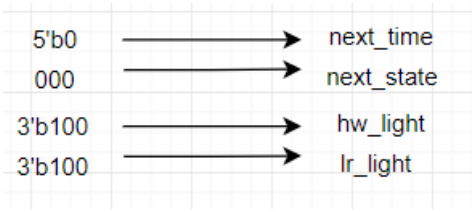


State C　　State D

下一個是 state D，代表 HW 是紅燈而 LR 是綠燈，output hw_light 為 3’b100，lr_light 為 3’b001。在此 state 時，times 會數 30 個週期，若是時間還沒數到 29，next_state 一樣維持在 state_D，times 繼續往上數，若 times 數到 29， next_state 就會轉換到 state E，next_times 歸零，右上為 state_D 的邏輯電路圖。

下一個是 state E，代表 HW 是紅燈而 LR 是黃燈，output hw_light 為 3’b100，lr_light 為 3’b010。在此 state 時，若是時間還沒數到 9，next_state 一樣維持在 state_E，times 繼續往上數，若 times 數到 9， next_state 就會轉換到 state F，next_times 歸零，左下為 state_E 的邏輯電路圖。





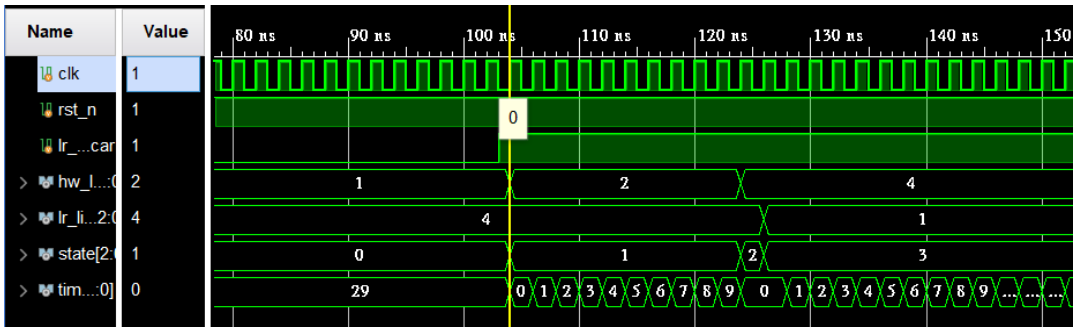
State E　　State F

下一個 state 為 state F，代表 HW 和 LR 都是紅燈，output hw_light 及 lr_light 皆為 3’b100。在此 state 時，經過一個 cycle 就會跳回 state A，next_time 設為 0，左下為 state F 的邏輯電路圖。

Verification

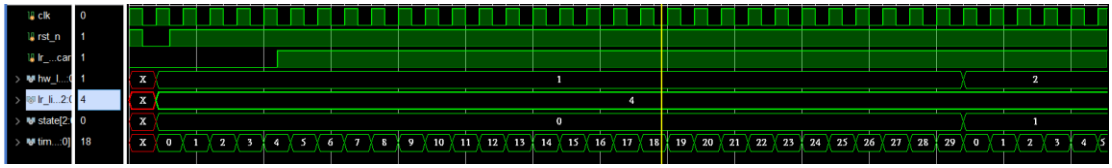
設計 testbench 來跑 verilog 的模擬圖，我們在 testbench 裡面順便將 time 及 state 顯示出來，觀察 times 及 state 波型的結果是否為我們所預期的

由於只有 State A 會受到 input lr_has_car 影響，因此特別針對 State A 做側資的測試，第一種情況是 times 已經數了三十個週期，但車子還沒來，State 就會停留在 A，直到偵測到車子來之後，才會進入到下一個 state。



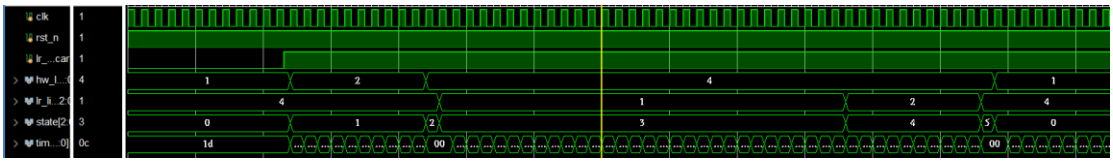
此結果為正確

接著是車子一直都在，times 一數到 29，state 就能轉換到 B

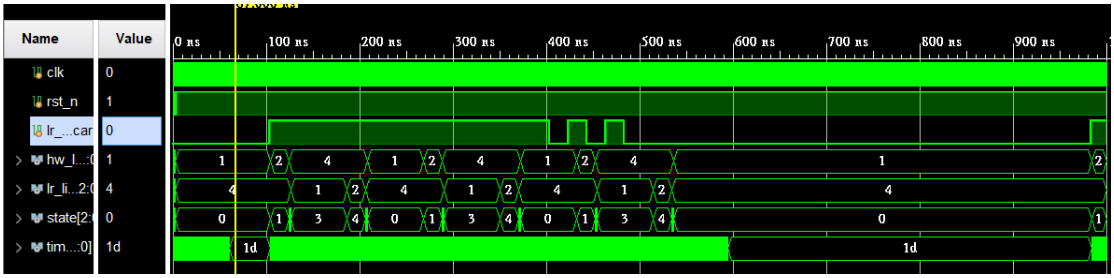


此結果為正確

最後驗證在其他的 state 時會不會受到 input 干擾，及在每個 state 倒數的時間是不是正確的



此結果為正確(state 轉換正確、倒數的時間秒數正確)



此結果為正確(其他 state 不會受到 input 影響)

綜合以上，我們可以說此邏輯電路為正確

FPGA demonstration

(一)State Transition Diagram

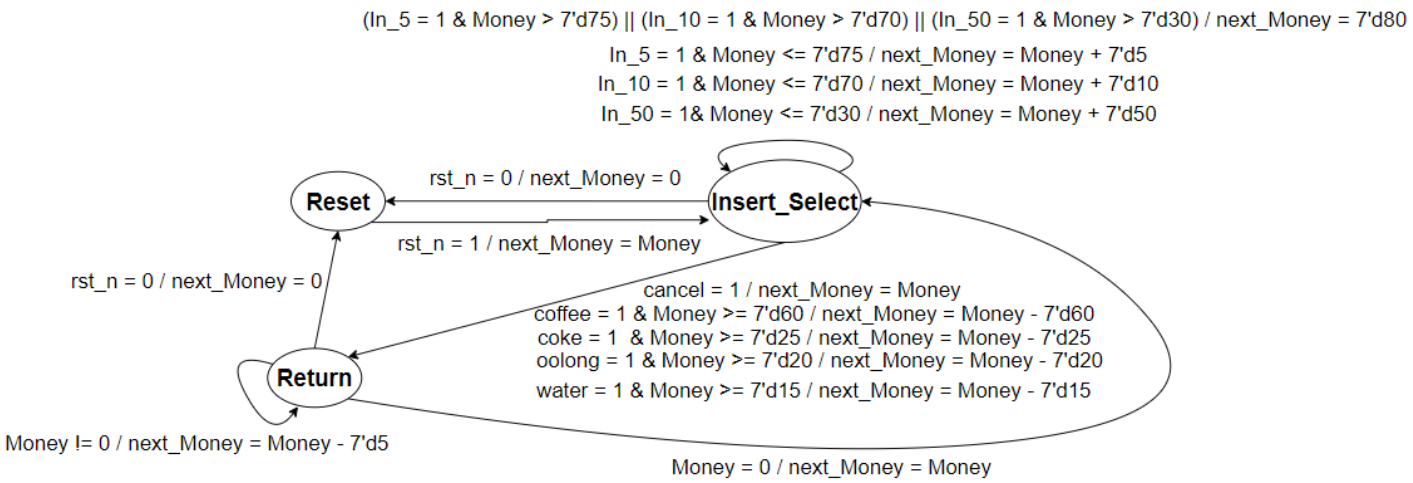
首先，首先先定義好在每一個 state，然後再畫出此題的 State Transition Diagram，如下：

State：Reset, Insert_Select, Return;

Input：In_5, In_10, In50；

Input：clk, rst_n, cancel;

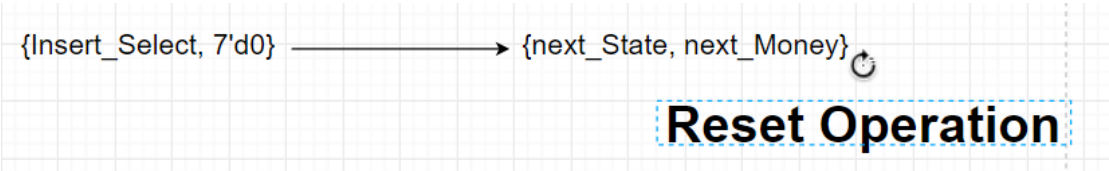
Input：coffee. coke, oolong, water;
Output：[7-1:0] Money；



State Transition Diagram

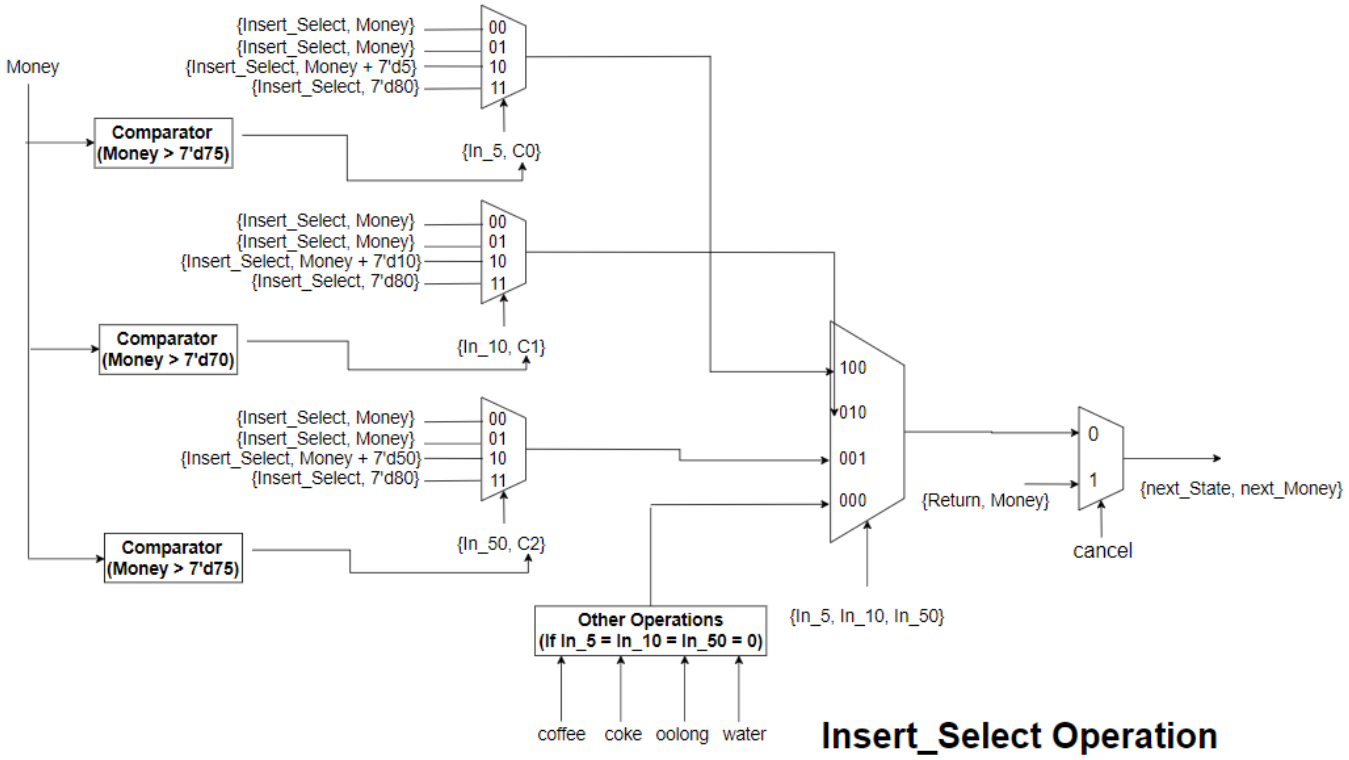
(二)Circuit Design and explanation：

1.當一進入 Reset 時，Money 會歸零，然後隨即進入 Insert_Select。

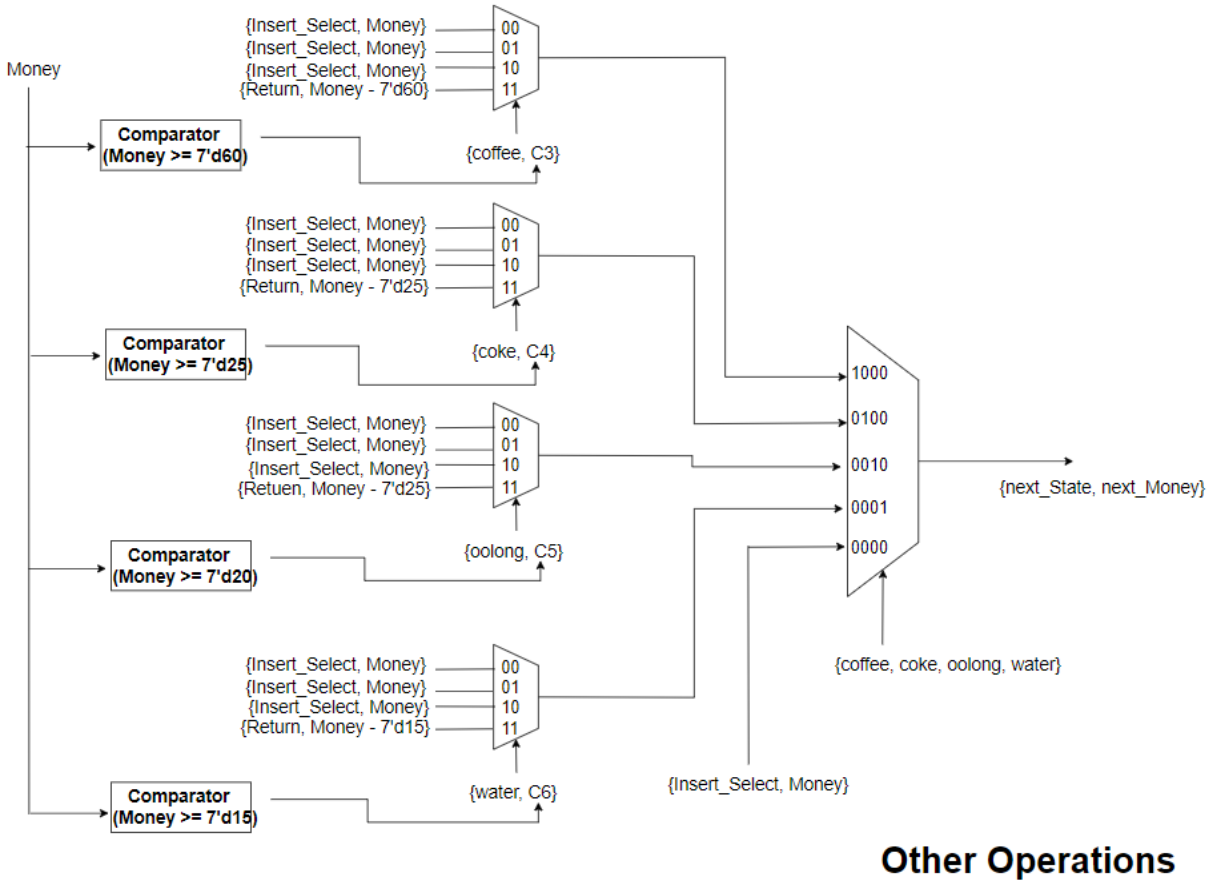


2.當進入 Insert_Select 時，首先要先判斷是否按下 cancel，如果 cancel = 1，next_State = Return；如果 cancel != 1，接著判斷是否有按下 In_5, In_10 or In_50，如果有其中一個按鈕被按下，接下來要判斷當下的金額是否超過 80(假設 In_10 被按下，如果 Money + 7'd10 > 7'd80 (或 Money > 7’d70)那麼 next_Money = 7'd80；如果 Money + 7'd10 <= 7'd80，next_Money = Money +

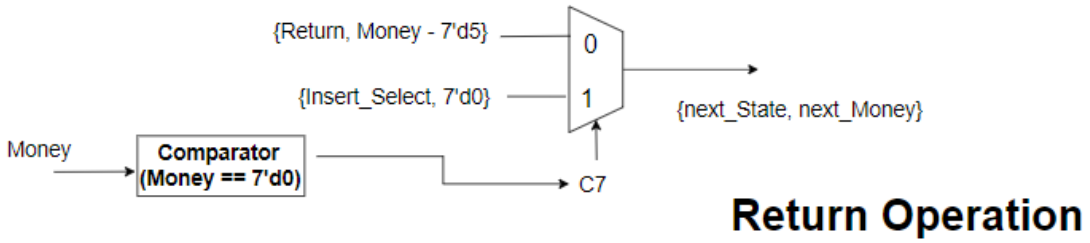
7’d10)。Insert_Select Operation 電路邏輯圖如下：



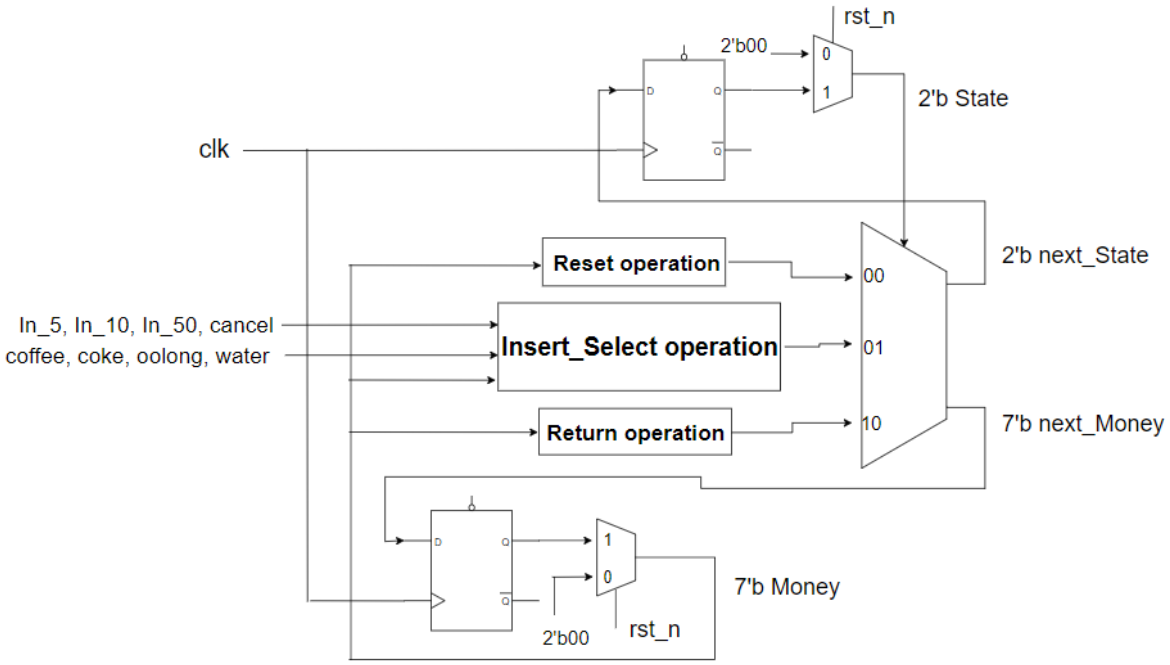
接著，如果鍵盤的按鈕(a,s,d,f)被按下(在此分別表示為 coffee, coke, oolong, water)，要判斷當下的金額夠不夠買，假設 a 被按下，也就是 coffee = 1，此時如果 Money >= 7'd60，則_next_Money = Money – 7’d60, next_State = Return；如果 Money < 7’d60，則維持不動。(這個時候可以對此判斷 LED 燈是否為亮)
如果沒有按鈕或是鍵盤的按鍵被按下，則 next_Money = Money, next_State = State(保持原本的狀態)。此部分為 Other Operations，電路邏輯圖如下：



3.當進入 Return 時，如果 Money != 7’d0，則 next_Money = Money - 7'd5, next_State = State；若 Money = 7’d0，則 next_Money = Money, next_State = Insert_Select(跳回 Insert_Select State)。Return Operation 電路邏輯圖如下：



以下是完整的邏輯電路圖：



Vending Machine

(三)FPGA Demonstration

接下來我們要將 verilog code 跑在板子上面，此分為兩個重要的環節：

(a)顯示：

1.decoder :為了將數字轉換為阿拉伯數字圖形，必須設計一個 SevenSegs decoder，當要表示一個數字時，會有對應的一組控制七段顯示器 的訊號輸出。

2.七段顯示器控制 ： 為了讓每個七段顯示器輪流亮燈，用 1-bit (因為只有兩位的七段顯示器會亮，其他位數皆是暗的)的 sctl_clk 訊號接一個 SevenSegs decoder 控制(4-bit 的 output(digit_display))，一個時間只會有一個七段顯示器亮(ex: 當 sctl_clk = 1'b0, digit_display = 4'b1110(只亮 digit0,其他皆暗掉), 當 sctl_clk = 1'b1, digit_display = 4'b1101(只亮 digit1,其他皆暗掉))。

(b)訊號處理：

1.Clock Divider： 首先我們要將石英震盪器的頻率除頻至人眼可已看得到的頻率，首先先做一個 24-bit 的加法器，最高位的那一項輸出(num[23])作為 Vending Machine 的頻率，此外我們也輸出了第 15 個 bit 作為輪流顯示七段顯示器的控制訊號(sctl_clk = num[15])。

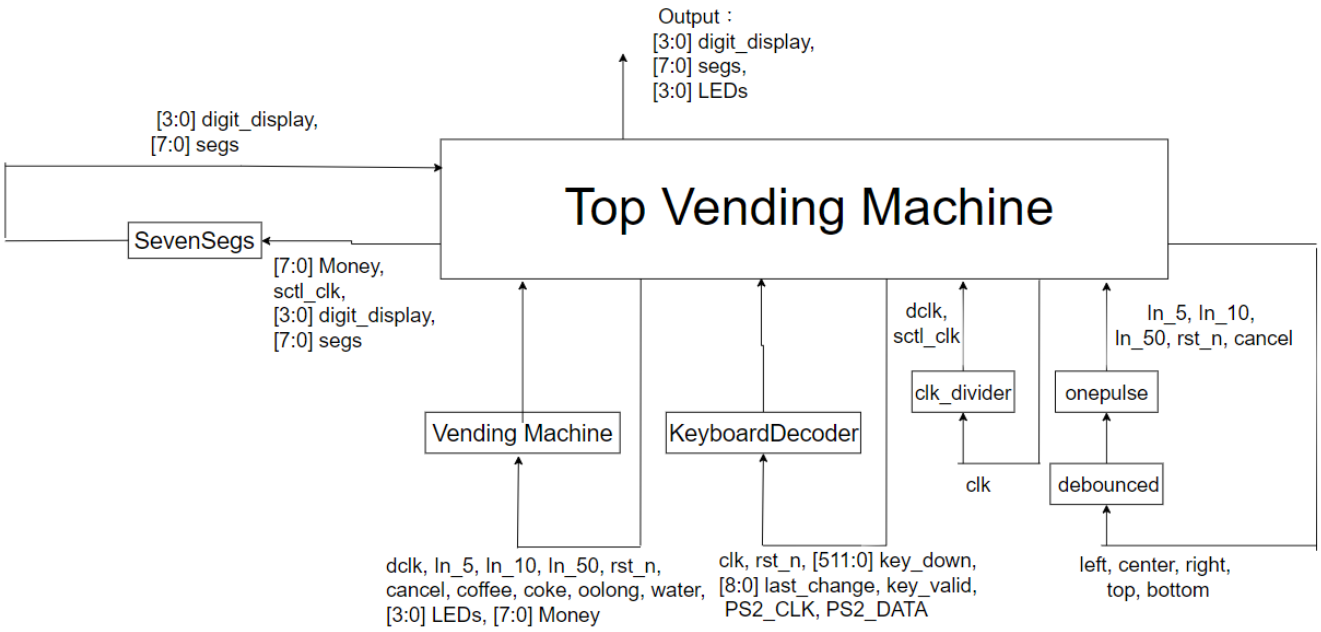
2.debounce： 由於用按壓按鈕的初期，按鈕會震盪而產生雜訊，為了過濾訊號，接了若干組 DFF，當每個 DFF 的值皆為 1 時，才輸出 1。

3.onepulse： 在 left, right, center, top, bottom 五個按鈕作用的時候，我們只需要效果持續一個週期，因此在 debounce 完之後，會接一個反向器做 AND，讓 left, right, center, top, bottom 的值為 1 的時間僅有 1 個週期。

4.KeyboardDecoder： 為了使鍵盤上的 a, s, d, f 按鍵能夠成為七段顯示器的 Input 之一，首先要先 import KeyboardDecoder，並將 KeyboardDecoder 所該使用到的 module 一併 copy 進來，接著在 Top Vending Machine 中多宣告四個 wire，分別是 coffee, coke, oolong, water，並搭配著 key_valid & key_down 的邏輯關係可以傳送鍵盤的訊號，例如 ：assign coffee = (key_valid == 1'b1 && key_down[9'b0_0001_1100])，此邏輯可以解釋成如果有按鍵按下且按下去的按鍵如果是 a，則 coffee = 1'b1，其他的按鍵以此類推。

(c)電路檢測方式：看 FPGA 板上所顯示的情形是否符合 specification。

以下是此題的 FPGA 的 Block Diagram：



心得

我覺得這次的 Lab5 老實說沒有到很難，但是因為有用到新的東西 (Keyboard)，所以花了不少時間去摸索。雖然花了蠻多時間的，但是至少學會用

keyboard 當作 Input 輸入。最近上課都在教如何使用新的 device 控制 FPGA 或是當作其他的 input 來源，我覺得相當有趣且新奇，不過蠻希望教授或是助教使用的用法可以再講詳細一點，尤其 Sample Code 希望可以講詳細一點，非常感謝！(107060011 涂皓鈞)

這次打報告的時間遠遠超出打 verilog 的時間，以後要訓練如何在短時間內畫好邏輯電路圖及如何在短時間內把腦袋中的構思清晰而簡短的表達出來。(107060015)

小組分工：

涂皓鈞–負責 Question2 以及 FPGA 題 verilog 以及對應題號的結報內容 + 心得

陳弘軽–負責 Question1, 3 的 verilog 以及對應題號的結報內容 + 心得